



日本国特許庁
PATENT OFFICE
JAPANESE GOVERNMENT

J. W. PRICE 949/261.8433
Takakazu SHIOMI et al
NAKI-B082

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出願年月日
Date of Application:

2000年 5月15日

出願番号
Application Number:

特願2000-141493

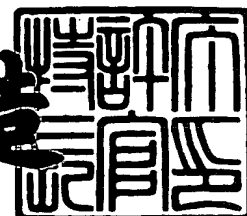
出願人
Applicant(s):

松下電器産業株式会社

2001年 1月19日

特許庁長官
Commissioner,
Patent Office

及川耕造



出証番号 出証特2000-3112897

【書類名】 特許願

【整理番号】 2022520224

【提出日】 平成12年 5月15日

【あて先】 特許庁長官殿

【国際特許分類】 G06F 9/46

【発明者】

【住所又は居所】 大阪府門真市大字門真 1 0 0 6 番地 松下電器産業株式会社内

【氏名】 塩見 隆一

【発明者】

【住所又は居所】 広島県東広島市鏡山 3 丁目 1 0 番 1 8 号 株式会社松下電器情報システム広島研究所内

【氏名】 久保岡 祐子

【特許出願人】

【識別番号】 000005821

【氏名又は名称】 松下電器産業株式会社

【代理人】

【識別番号】 100097445

【弁理士】

【氏名又は名称】 岩橋 文雄

【選任した代理人】

【識別番号】 100103355

【弁理士】

【氏名又は名称】 坂口 智康

【選任した代理人】

【識別番号】 100109667

【弁理士】

【氏名又は名称】 内藤 浩樹

【手数料の表示】

【予納台帳番号】 011305

【納付金額】 21,000円

【提出物件の目録】

【物件名】 明細書 1

【物件名】 図面 1

【物件名】 要約書 1

【包括委任状番号】 9809938

【書類名】 明細書

【発明の名称】 J a v a アプリケーション実行装置

【特許請求の範囲】

【請求項 1】 複数のスレッドで構成されるタスク単位で資源回収を行う O S と、

J a v a アプリケーションを記憶するアプリケーション記憶手段と、

前記アプリケーション記憶手段が記憶する J a v a アプリケーションを実行する J a v a V i r t u a l M a c h i n e と、

前記 J a v a アプリケーションを前記 O S が生成するタスク及びスレッドと組にして管理し、終了するアプリケーションが使用している資源を前記 O S に回収する指示を出すアプリケーション管理手段と、

前記 J a v a アプリケーション毎にスレッドを生成し、生成したスレッド上で前記アプリケーションのメソッドを呼び出すクラスライブラリを備えることを特徴とする J a v a アプリケーション実行装置。

【請求項 2】 前記 J a v a V i r t u a l M a c h i n e は、タスク・スレッドを生成し、生成したスレッド上で J a v a V i r t u a l M a c h i n e が使用する資源を確保することを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 3】 前記アプリケーション管理手段は、スレッドによって J a v a アプリケーションを特定し、

前記クラスライブラリは、前記 J a v a アプリケーションに対して共通のスレッド生成し管理することを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 4】 前記アプリケーション管理手段は、J a v a アプリケーションの終了を前記クラスライブラリに通知し、

前記クラスライブラリは、アプリケーション終了時に保持している J a v a アプリケーションに関するデータを削除することを特徴とする請求項 3 記載の J a v a アプリケーション実行装置。

【請求項 5】 前記 O S は、前記クラスライブラリが J a v a アプリケーション

ンを呼び出すタイミングを通知し、

前記クラスライブラリは、生成するスレッドに対応してキューを生成し、前記 OS からの通知によって、J a v a アプリケーションに通知する情報を前記キューに代入し、前記生成されたスレッドは前記キューに代入された情報を取り出し、J a v a アプリケーションを呼び出すことを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 6】 前記クラスライブラリは、J a v a アプリケーションからのリスナー登録時に前記 J a v a アプリケーションに対応するスレッドを生成することを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 7】 前記クラスライブラリは、J a v a アプリケーションが C o m p o n e n t クラスのインスタンスを生成する時に前記 J a v a アプリケーションに対応するスレッドを生成することを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 8】 前記クラスライブラリは、J a v a アプリケーションの起動時に前記 J a v a アプリケーションに対応するスレッドを生成することを特徴とする請求項 1 記載の J a v a アプリケーション実行装置。

【請求項 9】 コンピュータ読み取り可能な記録媒体であって、
複数のスレッドで構成されるタスク単位で資源回収を行う OS と、
J a v a アプリケーションを記憶するアプリケーション記憶手段と、
前記アプリケーション記憶手段が記憶する J a v a アプリケーションを実行する J a v a V i r t u a l M a c h i n e と、

前記 J a v a アプリケーションを前記 OS が生成するタスク及びスレッドと組にして管理し、終了するアプリケーションが使用している資源を前記 OS に回収する指示を出すアプリケーション管理手段と、

前記 J a v a アプリケーション毎にスレッドを生成し、生成したスレッド上で前記アプリケーションのメソッドを呼び出すクラスライブラリの各機能を発揮するプログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】

本発明は、J a v a アプリケーションが終了時に解放しなかった資源の回収機能を備えるJ a v a アプリケーション実行装置に関する。

【0 0 0 2】

【従来の技術】

従来のアプリケーション実行装置における資源回収方法は、特開平 8 - 1 2 3 7 0 0 「資源管理方法」に記述されている。アプリケーションはタスクに割り当てられ、タスクは複数のスレッドを有しアプリケーションを実行する。アプリケーションが使用する資源は、スレッドに関するものはスレッド単位で管理され、タスクに関する物はタスク単位で管理され、システム全体に関する物は、システム全体で管理される。図 1 1 は、これら管理状態を図示したものである。空間 1 1 0 1 はシステム全体を表し、システム全体に関する資源は空間資源 1 1 0 4 で管理される。システム中には複数のタスク 1 1 0 2 が存在し、タスクに関する資源は、タスク資源 1 1 0 5 で管理される。タスク 1 1 0 2 は複数のスレッド 1 1 0 3 を有し、スレッドに関する資源はスレッド資源 1 1 0 6 で管理される。管理されている資源は、スレッド終了時にはスレッドに関する資源だけが回収され、タスク終了時にはタスクに関する資源とその中に含まれるスレッドに関する資源が回収される。

【0 0 0 3】

一方、J a v a アプリケーションを実行するJ a v a ミドルウェアは、様々なOS上に実現されている。現在、Sun M i c r o s y s t e m s 社から提供されている、J a v a ミドルウェアは、J a v a アプリケーションの起動と同時に起動され、J a v a アプリケーションの終了と共に終了する。J a v a ミドルウェア自体でJ a v a アプリケーションが使用した資源回収は行わず、J a v a ミドルウェアが終了することで、OSによる資源回収が行われている。

【0 0 0 4】

【発明が解決しようとする課題】

一般にJ a v a ミドルウェアの起動処理は、最初に多くのクラスをメモリーに読み込むため時間がかかる。よって、J a v a アプリケーションのみを実行する

システムで、アプリケーションの起動、終了毎にミドルウェアの終了・再起動を行うと切り替え時間が長くなる。ミドルウェアを再起動せずに次のアプリケーションを実行すると資源回収が不十分なため、資源不足でアプリケーションが動作しなくなるという問題点がある。また、J a v a ミドルウェアは、複数の J a v a アプリケーションに対して共通のスレッドを用意し、そのスレッドでアプリケーションから登録されたリスナー呼び出しやアプリケーションの描画メソッド呼び出しを行う。この共通スレッドで実行される J a v a アプリケーションのコード中でシステム資源を確保し使用する場合があるため、O S レベルでのスレッド単位での資源管理が不可能という問題もある。

【 0 0 0 5 】

本発明は、J a v a ミドルウェアにおいてアプリケーション毎に専用スレッドを生成しリスナー呼び出しやアプリケーションの描画メソッド呼び出しを行うことにより、O S が備えるタスク単位の資源回収機能を利用した資源回収機能を実現し、再起動なしで長時間安定動作する J a v a ミドルウェアを搭載した J a v a アプリケーション実行装置を提供する。

【 0 0 0 6 】

【課題を解決するための手段】

上記課題を解決するため本発明は、複数のスレッドで構成されるタスク単位で資源回収を行う O S と、J a v a アプリケーションを記憶するアプリケーション記憶手段と、前記アプリケーション記憶手段が記憶する J a v a アプリケーションを実行する J a v a V i r t u a l M a c h i n e と、前記 J a v a アプリケーションを前記 O S が生成するタスク及びスレッドと組にして管理し、終了するアプリケーションが使用している資源を前記 O S に回収する指示を出すアプリケーション管理手段と、前記 J a v a アプリケーション毎にスレッドを生成し、生成したスレッド上で前記アプリケーションのメソッドを呼び出すクラスライブラリを備えることとしている。

【 0 0 0 7 】

【発明の実施の形態】

以下、本発明に係るアプリケーション実行装置の具体的な実施の形態を、図面

を参照しながら説明する。

【0008】

(実施の形態1)

図1は、本発明に係るJ a v aアプリケーション実行装置の形態を表したものであり、アプリケーション入力部101、アプリケーション記憶部102、J a v aミドルウェア103、OS104、ハードウェア105、制御部106で構成される。

【0009】

アプリケーション入力部101は、実行すべきJ a v aアプリケーションを受け付ける。具体的にはフロッピーディスクやCD等のメディアを通してJ a v aアプリケーションが供給される場合、本構成要素はフロッピーディスクドライブ、CDドライブ等で構成される。またネットワークや放送局からの電波をを通してJ a v aアプリケーションが配信される場合は、ネットワークインターフェースボードや放送受信機器で構成される。内蔵されるJ a v aアプリケーションのみを実行する装置においては、アプリケーション記憶部102にJ a v aアプリケーションを記憶しておくだけで良いので、本要素は不要となる。

【0010】

アプリケーション記憶部102は、実行すべきJ a v aアプリケーションを記憶する。具体的には、RAM、ROM、ハードディスクなどの内蔵機器や、CDドライブ、フロッピーディスクや、フロッピーディスクなどのリムーバブルメディア等で構成される。

【0011】

J a v aミドルウェア103は、アプリケーション記憶部102が記憶するJ a v aアプリケーションを実行する。内部には、VM部103a、アプリケーション管理部103b、クラスライブラリ格納部103cを構成要素として持つ。

【0012】

VM部103aはアプリケーション記憶部102が記憶するJ a v aアプリケーションを実行する。J a v aアプリケーションを構成するバイトコードを逐次解析し実行する。J a v aアプリケーションは、システムが備えているデバイス

を制御するため、デバイスを制御するためのクラスライブラリを呼び出す。VM部103aは、Javaアプリケーションがクラスライブラリを呼び出すとき、クラスライブラリ格納部103cが格納するクラスライブラリから必要なものを呼び出し実行する。また、Javaアプリケーションはスレッドを生成したりオブジェクトのインスタンスを生成するためメモリーを使用する。この場合、VM部103aは後述するOS104のカーネル104aを呼び出し、スレッドを生成しメモリーを確保する。

【0013】

アプリケーション管理部103bは、VM部103aが実行するJavaアプリケーションを管理し、以下の3つの機能を有する。

【0014】

Javaアプリケーションに対応するタスクとタスク中の最初のスレッドOS104のカーネル104aに依頼して作成し、VM部103aに依頼して、そのスレッド上でJavaアプリケーションを実行する。この際、アプリケーションとタスクを対応付けて保持する。また、動作しているJavaアプリケーションから派生して生成される全てのスレッドを管理する。

【0015】

図2は、アプリケーション管理部103bがアプリケーションとタスクとスレッドを対応付けて管理しているテーブルの例である。図2(1)では2つのアプリケーションが存在し、アプリケーション管理部103bは、2つのアプリケーションにアプリケーションID「1」、「2」を割り当てている。各アプリケーションにはOSによって用意されたタスクのタスクID「201」「202」が保持されている。アプリケーションID「1」のアプリケーションは2つのスレッドで実行されており、それらのスレッドのスレッドID「1」「2」を保持している。アプリケーションID「2」のアプリケーションは3つのスレッドで実行されており、それらのスレッドのスレッドID「4」「5」「6」を保持している。ここで、アプリケーションID「1」のアプリケーションがスレッドを生成すると、追加されたスレッドのスレッドIDがテーブルに追加される。図2(2)は、生成されたスレッドのスレッドID「7」が追加された状態を表す。ま

た、図2(2)の状態、スレッドID「5」のスレッドがアプリケーションにより削除されると、図2(3)のようテーブルからスレッドID「5」が削除される。

【0016】

アプリケーションが終了する際、カーネル104aに終了するアプリケーションを実行しているタスクを通知し、OS104による資源回収を依頼する。またクラスライブラリ格納部103cが格納するクラスライブラリにアプリケーションが終了することを通知する。その後、アプリケーションを実行しているタスク及びその中に含まれる全てのスレッドを終了し、VM部103aによる実行を停止する。

【0017】

例えば、アプリケーション管理部103bが図2(3)の状態、アプリケーションID「2」のアプリケーションを終了させる場合、まずカーネル104aに対応するタスクID「202」を通知し、アプリケーションに関する資源回収を依頼する。資源回収が終了したのち、アプリケーションに対応するタスクID「202」noタスク及びその中に含まれるスレッドID「4」,「6」のスレッドを終了させる。

【0018】

クラスライブラリが呼び出された際、クラスライブラリがどのアプリケーションから呼び出されているかをアプリケーション管理部103bに問い合わせる。クラスライブラリも、アプリケーションを実行しているスレッドまたは、そのスレッドから派生して生成されたスレッドで実行されているので、アプリケーション管理部103bは問い合わせ自体が実行されているスレッドからアプリケーションを特定しクラスライブラリに通知する。

【0019】

例えば、アプリケーション管理部103bが図2(3)の状態、クラスライブラリから呼び出されたとき、アプリケーション管理部103bは、その呼び出しを実行しているスレッドのスレッドIDを取得する。ここでスレッドIDは、オブジェクトのハッシュコードwo用いるとすると、呼び出しで使用されたスレ

ッドのスレッドクラスのインスタンスを取得し、ハッシュコードを返す関数 `hashCode` によって取得することができる。次に取得したスレッド ID を図 2 (3) のテーブルデータの中から検索し、アプリケーション ID 「2」 に対応付けられて保持していることが判明するので、このアプリケーション ID 「2」 を返す。

【0020】

クラスライブラリ格納部 103c は、システムが備えるデバイスを制御するためのクラスライブラリを格納し、具体的には ROM やハードディスクなどで構成される。クラスライブラリ格納部 103c は、図中の「ライブラリ 1」「ライブラリ 2」、のように、一般に複数のクラスライブラリを保持する。各クラスライブラリは OS 104 のライブラリ格納部 104b が格納するライブラリを呼び出し、システムが備えるデバイスを制御する。

【0021】

クラスライブラリは、アプリケーションからリスナーと呼ばれるコールバック関数を登録され、これを特定のタイミングで呼び出す。クラスライブラリーは、リスナー登録される際、リスナー登録するアプリケーションに対して専用のスレッドを作成する。リスナー登録は Java アプリケーションのスレッドで実行されているので、このスレッド上で専用スレッドを作成すれば、自動的にアプリケーションのタスクのスレッドに組み込まれる。このスレッドはアプリケーション管理部 103b でアプリケーションに対応付けられ管理される。リスナー呼び出し時は、この予め作成しておいたスレッドでリスナーを呼び出す。予め作成したスレッドを用いてリスナーを呼び出す具体的な方法の詳細は後述する。

【0022】

ここで、1つのアプリケーションが複数回リスナー登録をする可能性がある。多くのスレッドを作りすぎないために、クラスライブラリはアプリケーション管理部 103b に問い合わせることで、アプリケーションを特定し、アプリケーションに対して1つで専用スレッドだけを作成することが出来る。これを実現するためにクラスライブラリはアプリケーションと専用スレッドと登録されたリスナーを対応付けて保持する。

【 0 0 2 3 】

図 3 は、クラスライブラリが保持するアプリケーションと専用スレッドと登録されたリスナーの組の 1 例である。(1) では、既にアプリケーション ID 「1」のアプリケーションからリスナー「L1」と「L4」が登録され、これらのリスナーを呼び出すために、専用スレッド「t h r e a d 1 0」がアプリケーション ID 「1」のアプリケーションから派生したスレッドとして生成されていることがわかる。ここで、新規にリスナー「L6」が登録されたとすると、クラスライブラリはアプリケーション管理部 1 0 3 b からアプリケーションのアプリケーション ID を取得する。ここで「2」が得られたとする。既に保持しているアプリケーション ID と比較し一致すれば、登録されるリスナーを登録するが、図 3 (1) を参照して一致するものはないので、新規に専用スレッドを作成しアプリケーション ID と登録されたリスナーとともに保持する。図 3 (2) はこの状態を表す。

【 0 0 2 4 】

クラスライブラリ格納部 1 0 3 c が格納するクラスライブラリはアプリケーションの終了を通知される。この時、終了するアプリケーションのリスナー登録情報を削除する。

【 0 0 2 5 】

例えば図 3 (2) の状態で、アプリケーション ID 「1」のアプリケーションの終了が通知されると、このアプリケーション ID 「1」の情報を削除し図 3 (3) のようになる。

【 0 0 2 6 】

次に、予め作成したスレッドを用いてリスナーを呼び出す具体的な方法を図 4 を参照しながら説明する。一般にクラスライブラリに登録されたリスナーはシステムの状態が変化したときなどに呼び出される。ユーザーからのキー入力時に呼び出し等は典型的な例である。リスナーを呼び出すタイミングは一般にシステムの変化を知っている OS 1 0 4 のライブラリ格納部 1 0 4 b に格納されるクラスライブラリから通知される。この通知は、OS 1 0 4 で生成されたタスク・スレッドで行われる。今、クラスライブラリが図 3 (2) の状態で 2 つの専用スレ

ド生成しているとする。図4の中で、401は、専用スレッド「thread10」を、402は専用スレッド「thread20」を、403はクラスライブラリに通知を行うOS104で生成されたスレッドを表している。アプリケーション用に生成された各専用スレッドに対してキュー411、412が用意される。クラスライブラリは、OS104のライブラリから通知された情報をキュー411、412に代入する。破線421、422は代入することを表し、431、432は代入された情報を表す。各専用スレッド401、402は、キュー411、412を監視し、キューの情報を発見すると、その情報を取り出し対応するリスナーを呼び出す。破線441、442はキューに代入された情報の取り出しを表す。図3(2)を参照して、専用スレッド401は、リスナー「L1」「L2」を呼び出し、専用スレッド402は、リスナー「L6」を呼び出す。

【0027】

OS104は、Javaミドルウェアを動作させることにより、Javaアプリケーションを間接的に動作させ、カーネル104aとライブラリ格納部104bが格納するライブラリで構成される。

【0028】

カーネル104aは、Javaミドルウェア103のアプリケーション管理部103bから依頼され、Javaアプリケーションを実行するタスクとスレッドを生成する。

【0029】

カーネル104aは、複数のスレッドを有するタスク単位で資源を管理する。スレッド上で実行されるアプリケーションから生成されるスレッドは、同じタスクのスレッドとして管理し、各スレッド上で実行されるアプリケーションによって確保される資源は、スレッドを含むタスクと対応付けて管理される。実際の資源の管理は、カーネル104aが行う場合と、資源を直接供給するライブラリ格納部104bが格納するライブラリが行う場合の2通りがある。

【0030】

また、カーネル104aは、Javaミドルウェア103のアプリケーション管理部103bからJavaアプリケーションを終了するため、Javaアプリ

ケーションに対応するタスクの通知を受けると、タスクに対応付けられて管理されている資源を全て回収する。

【 0 0 3 1 】

ライブラリ格納部 1 0 4 b は、システムが備えるデバイスを制御するためのライブラリを格納し、具体的には ROM やハードディスクなどで構成される。ライブラリ格納部 1 0 4 b は、図中の「ライブラリ 1」「ライブラリ 2」、のよう
に、一般に複数のライブラリを保持する。各ライブラリはハードウェア 1 0 5 の
デバイスを制御する。

【 0 0 3 2 】

また、ライブラリは資源を供給すると共に、使用されている資源をタスク単位
で管理する場合もある。この場合、資源供給の際にカーネル 1 0 4 a n i 問
い合わせ、資源を供給するタスクを問い合わせタスクと資源を組にして管理を行う。
カーネル 1 0 4 a より特定のタスクに関する資源回収の指示が有ると、指示され
たタスクに対応する資源を回収する。

【 0 0 3 3 】

ハードウェア 1 0 5 は、OS 1 0 4 や J a v a ミドルウェア 1 0 3 を動作させる
CPU 1 0 5 a 及びデバイス 1 0 5 b で構成される。

【 0 0 3 4 】

CPU 1 0 5 は、ソフトウェアをカーネル 1 0 4 a、ライブラリ格納部 1 0 4
b が格納するライブラリ、VM部 1 0 3 a、クラスライブラリ格納部 1 0 3 b が
格納するクラスライブラリ等のプログラムコードを実行する。

【 0 0 3 5 】

デバイス 1 0 5 b は、具体的には、メモリーデバイス、ディスプレイデバイス
、入力デバイス（キーボード、リモコン等）、ファイルシステム、ネットワーク
デバイスなどが含まれ、それぞれのデバイスはクラスライブラリ格納部 1 0 3 c
が格納するクラスライブラリ及びライブラリ格納部 1 0 4 b が格納するライブラ
リと通して、J a v a アプリケーションから制御可能である。

【 0 0 3 6 】

制御部 1 0 6 は、システム上で動作するアプリケーションの開始、終了などを

指示する。

【 0 0 3 7 】

アプリケーションの開始は、アプリケーション入力部 1 0 1 に、アプリケーションを受け付け、アプリケーション記憶部 1 0 2 に記憶する指示を行い、その後、アプリケーション管理部 1 0 3 b にアプリケーションを実行する指示を出す。

【 0 0 3 8 】

アプリケーションの終了は、アプリケーション管理部 1 0 3 b に実行しているアプリケーションの終了を指示する。アプリケーション管理部 1 0 3 b は、VM 部 1 0 3 a にアプリケーションの実行の停止を指示する。また、OS 1 0 4 のカーネル 1 0 4 a にアプリケーションに対応するタスクを終了し、このタスクを構成するスレッドで確保された資源を全て回収する指示を与える。

【 0 0 3 9 】

図 5 は、本実施の形態のアプリケーション実行装置において、J a v a アプリケーションがクラスライブラリにリスナー登録する際のクラスライブラリの動作を説明するフローチャートである。

【 0 0 4 0 】

クラスライブラリは、リスナー登録のためのメソッドを呼ばれると、まずアプリケーション管理部 1 0 3 b からアプリケーションを特定するアプリケーション ID を取得する（ステップ S 5 0 1）。次に、リスナー、専用スレッド、アプリケーション ID を組にした登録情報の 1 つを取り出し（ステップ S 5 0 2）、登録情報のアプリケーション ID とアプリケーション管理部 1 0 3 b から取得したアプリケーション ID を比較する（ステップ S 5 0 3）。一致すれば、登録情報に登録されたリスナーを追加登録する（ステップ S 5 0 4）。不一致の場合は、全登録情報に関して比較を行い（ステップ S 5 0 5）、一致する登録情報がなければ、アプリケーション専用スレッドを生成する（ステップ S 5 0 6）。その後、リスナー、生成した専用スレッド、アプリケーション ID を組にした登録情報を作成し保持する（ステップ S 5 0 7）。

【 0 0 4 1 】

図 6 と図 7 は、クラスライブラリが OS 部 1 0 4 のライブラリからリスナー呼

び出しのタイミングを通知された際の、クラスライブラリのリスナー呼び出しの動作を説明するフローチャートである。

【 0 0 4 2 】

図 6 は、OS 部 1 0 4 で生成されたライブラリからのスレッドで実行されるクラスライブラリの動作を表すフローチャートである。クラスライブラリは、OS 部 1 0 4 のライブラリからリスナー呼び出しのタイミングを通知されると、リスナー、専用スレッド、アプリケーション ID を組にした登録情報の 1 つを取り出す（ステップ S 6 0 1）。登録情報の専用スレッドのキューにリスナー呼び出しに必要な呼出情報を代入する（ステップ S 6 0 2）。これを全登録情報に対して行う（ステップ S 6 0 3）。

【 0 0 4 3 】

図 7 は、専用スレッドで実行されるクラスライブラリの動作を表すフローチャートである。専用スレッドでは、キューに呼出情報が代入されていないかを監視している（ステップ S 7 0 1）。キューに呼出情報があれば、それを取り出す（ステップ S 7 0 2）。専用スレッドに対応するリスナーの 1 つを取り出し（ステップ S 7 0 3）、呼出情報に応じて登録されているリスナーを呼び出す（ステップ S 7 0 4）。これを全リスナーに対して行う（ステップ S 7 0 5）。

【 0 0 4 4 】

図 8 は、アプリケーション終了時のアプリケーション管理部 1 0 3 b の動作を説明するフローチャートである。

【 0 0 4 5 】

制御部 1 0 6 よりアプリケーション終了の指示を受けるとアプリケーション管理部 1 0 3 b は、カーネル 1 0 4 a にタスク ID を指定し資源回収を依頼する（ステップ S 8 0 1）。カーネル 1 0 4 a は OS 1 0 4 内のライブラリ格納部 1 0 4 b 内のライブラリを呼び出すなどして、資源回収を行う。次に、アプリケーション管理部 1 0 3 b はクラスライブラリ格納部 1 0 3 c 内の各クラスライブラリにアプリケーション ID を指定しアプリケーションの終了を通知する（ステップ S 8 0 2）。各クラスライブラリは、指定されたアプリケーション ID を含む、リスナー、専用スレッド、アプリケーション ID を組にした登録情報を削除する

。最後に、アプリケーション管理部 1 0 3 b はアプリケーションを実行しているタスクを終了させる。

【 0 0 4 6 】

なお、ステップ S 8 0 1 とステップ S 8 0 2 は順序が逆でも実施可能である。

【 0 0 4 7 】

実施の形態 1 において、VM 部 1 0 3 a は、アプリケーションを実行するスレッド上で実行され、その中で VM 部 1 0 3 a が動作するのに必要な資源も確保される。しかし、実施の形態 1 においては、これらの資源はアプリケーションの終了と共に回収されてしまうため、再度確保しなければならない。よって、システム起動時に VM 部 1 0 3 a の専用スレッドを起動し、このスレッド上で VM 部 1 0 3 a が必要な資源を確保することで、アプリケーション終了による資源再取得を不要にすることができる。

【 0 0 4 8 】

実施の形態 1 において、クラスライブラリ格納部 1 0 3 b が格納するクラスライブラリは、生成した J a v a アプリケーション用の専用スレッドでキューに呼び出し情報があるかどうかを監視している。しかし、監視を続けるとシステム全体の実行速度を下げる事になる。そこで、専用スレッドはウェイとしておき、OS 1 0 4 のライブラリからリスナー呼び出しのタイミングを通知されたとき、ウェイとしている専用スレッドを再起動してもよい。図 9、図 1 0 は、この場合のクラスライブラリの動作を表すフローチャートである。

【 0 0 4 9 】

図 9 は、OS 部 1 0 4 で生成されたライブラリからのスレッドで実行されるクラスライブラリの動作を表すフローチャートである。クラスライブラリは、OS 部 1 0 4 のライブラリからリスナー呼び出しのタイミングを通知されると、リスナー、専用スレッド、アプリケーション ID を組にした登録情報の 1 つを取り出す（ステップ S 6 0 1）。登録情報の専用スレッドのキューにリスナー呼び出しに必要な呼出情報を代入する（ステップ S 6 0 2）。その後、ウェイトしている専用スレッドを起動する（ステップ S 9 0 1）。これを全登録情報に対して行う（ステップ S 6 0 3）。

【 0 0 5 0 】

図 1 0 は、専用スレッドで実行されるクラスライブラリの動作を表すフローチャートである。専用スレッドは、起動されると即座にウェイト状態になる（ステップ S 1 0 0 1）。ウェイト状態から起動されると、キューに呼出情報が代入されているはずなので、それを取り出す（ステップ S 7 0 2）。専用スレッドに対応するリスナーの 1 つを取り出し（ステップ S 7 0 3）、呼出情報に応じて登録されているリスナーを呼び出す（ステップ S 7 0 4）。これを全リスナーに対して行う（ステップ S 7 0 5）。リスナー呼び出しが終了すると再びウェイト状態に入る。

【 0 0 5 1 】

システムから J a v a アプリケーションのメソッドを呼び出すのはリスナーだけでなく C o m p o n e n t クラスの p i a n t メソッドもある。この p i a n t メソッドもアプリケーションから派生するスレッドで呼び出すため、C o m p o n e n t 生成時に p a i n t メソッドを呼び出す専用スレッドを生成する。

【 0 0 5 2 】

実施の形態 1 では、リスナー呼び出し時、また、実施の形態の変形例その 3 では C o m p o n e n t 生成時にアプリケーションのスレッドから派生した専用スレッドを作成した。しかし、アプリケーションがほぼ確実にリスナー登録するクラスライブラリには、アプリケーション起動時に専用スレッドを起動しても良い。これにより、クラスライブラリはリスナー登録時に毎回専用スレッドの有無を検査する必要がなくなり短時間でリスナー登録処理を完了することが出来る。

【 0 0 5 3 】

【発明の効果】

以上説明したように、本発明によれば、複数のスレッドで構成されるタスク単位で資源回収を行う O S と、J a v a アプリケーションを記憶するアプリケーション記憶手段と、前記アプリケーション記憶手段が記憶する J a v a アプリケーションを実行する J a v a V i r t u a l M a c h i n e と、前記 J a v a アプリケーションを前記 O S が生成するタスク及びスレッドと組にして管理し、終了するアプリケーションが使用している資源を前記 O S に回収する指示を出す

アプリケーション管理手段と、前記 J a v a アプリケーション毎にスレッドを生成し、生成したスレッド上で前記アプリケーションのメソッドを呼び出すクラスライブラリを備えることとしている。

【 0 0 5 4 】

クラスライブラリは、アプリケーション毎にスレッドを生成することにより、J a v a ミドルウェアでの資源回収を不要にし、O S のみでのタスク・スレッドによる資源回収を可能にしている。これにより短時間で資源回収が可能になる。

【 0 0 5 5 】

更に、前記 J a v a V i r t u a l M a c h i n e は、タスク・スレッドを生成し、生成したスレッド上で J a v a V i r t u a l M a c h i n e が使用する資源を確保することとしている。

【 0 0 5 6 】

これにより J a v a アプリケーション終了時に J a v a V i r t u a l M a c h i n e が使用する資源が解放され、資源の再獲得をする処理を省くことが出来、アプリケーションの終了をより短かい時間で実行できる。

【 0 0 5 7 】

更に、前記アプリケーション管理手段は、スレッドによって J a v a アプリケーションを特定し、前記クラスライブラリは、前記 J a v a アプリケーションに対して共通のスレッド生成し管理することとしている。

【 0 0 5 8 】

これにより、生成するスレッド数を減らすことが出来、システム全体の J a v a アプリケーション実行効率の低下を防ぐ。

【 0 0 5 9 】

更に、前記アプリケーション管理手段は、J a v a アプリケーションの終了を前記クラスライブラリに通知し、前記クラスライブラリは、アプリケーション終了時に保持している J a v a アプリケーションに関するデータを削除することとしている。

【 0 0 6 0 】

クラスライブラリに余分な情報が残ることを防ぐことが出来る。

【0061】

更に、前記OSは、前記クラスライブラリがJavaアプリケーションを呼び出すタイミングを通知し、前記クラスライブラリは、生成するスレッドに対応してキューを生成し、前記OSからの通知によって、Javaアプリケーションに通知する情報を前記キューに代入し、前記生成されたスレッドは前記キューに代入された情報を取り出し、Javaアプリケーションを呼び出すこととしている。

【0062】

キューを使用することで、必要な情報を伝達し、アプリケーション用に生成されたスレッド上でJavaアプリケーションを呼び出すことが出来る。

【0063】

更に、前記クラスライブラリは、Javaアプリケーションからのリスナー登録時に前記Javaアプリケーションに対応するスレッドを生成することとしている。

【0064】

リスナー登録時にスレッドを生成することで、必要時のみスレッドを生成することが出来る。

【0065】

更に、前記クラスライブラリは、JavaアプリケーションがComponentクラスのインスタンスを生成する時に前記Javaアプリケーションに対応するスレッドを生成することとしている。

【0066】

Componentのインスタンス生成時にスレッドを生成することで、必要時のみスレッドを生成することが出来る。

【0067】

更に、前記クラスライブラリは、Javaアプリケーションの起動時に前記Javaアプリケーションに対応するスレッドを生成することとしている。

【0068】

高い確率でリスナー登録されるクラスライブラリではアプリケーション起動時

にスレッドを作成しておくことで、リスナー登録の処理を短時間で実行できる。

【0069】

ことを特徴とする J a v a アプリケーション実行装置。

【0070】

更に、コンピュータ読み取り可能な記録媒体であって、複数のスレッドで構成されるタスク単位で資源回収を行う O S と、 J a v a アプリケーションを記憶するアプリケーション記憶手段と、前記アプリケーション記憶手段が記憶する J a v a アプリケーションを実行する J a v a V i r t u a l M a c h i n e と、前記 J a v a アプリケーションを前記 O S が生成するタスク及びスレッドと組にして管理し、終了するアプリケーションが使用している資源を前記 O S に回収する指示を出すアプリケーション管理手段と、前記 J a v a アプリケーション毎にスレッドを生成し、生成したスレッド上で前記アプリケーションのメソッドを呼び出すクラスライブラリの各機能を発揮するプログラムとすることとしている。このような構成によって、資源回収機構を有するアプリケーション実行装置としてコンピュータを利用することができる。

【図面の簡単な説明】

【図 1】

本発明に係る O S による資源回収機能を有する J a v a アプリケーション実行装置の実施の形態 1 の構成図

【図 2】

(1) 実施の形態 1 に含まれるアプリケーション管理部 1 0 3 b が、アプリケーション I D、タスク I D、スレッド I D の組を管理している状態を表す図

(2) 実施の形態 1 に含まれるアプリケーション管理部 1 0 3 b が、アプリケーション I D、タスク I D、スレッド I D の組を管理している状態を表す図

(3) 実施の形態 1 に含まれるアプリケーション管理部 1 0 3 b が、アプリケーション I D、タスク I D、スレッド I D の組を管理している状態を表す図

【図 3】

(1) 実施の形態 1 において、クラスライブラリがアプリケーション I D、スレッドインスタンス、リスナーインスタンスの組を管理している状態を表す図

(2) 実施の形態 1 において、クラスライブラリがアプリケーション ID、スレッドインスタンス、リスナーインスタンスの組を管理している状態を表す図

(3) 実施の形態 1 において、クラスライブラリがアプリケーション ID、スレッドインスタンス、リスナーインスタンスの組を管理している状態を表す図

【図 4】

実施の形態 1 において、クラスライブラリが登録されたリスナーのメソッドを呼び出す手順と OS のスレッド、アプリケーション用に生成したスレッド及びそのキューの関係を示す図

【図 5】

実施の形態 1 において、クラスライブラリはアプリケーション専用スレッドを生成し管理する際の動作を表すフローチャート

【図 6】

実施の形態 1 において、クラスライブラリがリスナーを呼び出す際、OS のスレッドで実行する動作を表すフローチャート

【図 7】

実施の形態 1 において、クラスライブラリがリスナーを呼び出す際、アプリケーション専用スレッドで実行する動作を表すフローチャート

【図 8】

実施の形態 1 において、アプリケーション管理部 103b がアプリケーション終了時に行う資源回収の動作を表すフローチャート

【図 9】

上記実施の形態 1 において、クラスライブラリがリスナーを呼び出す際、OS のスレッドで実行する動作を表すフローチャート

【図 10】

上記実施の形態 1 において、クラスライブラリがリスナーを呼び出す際、アプリケーション専用スレッドで実行する動作を表すフローチャート

【図 11】

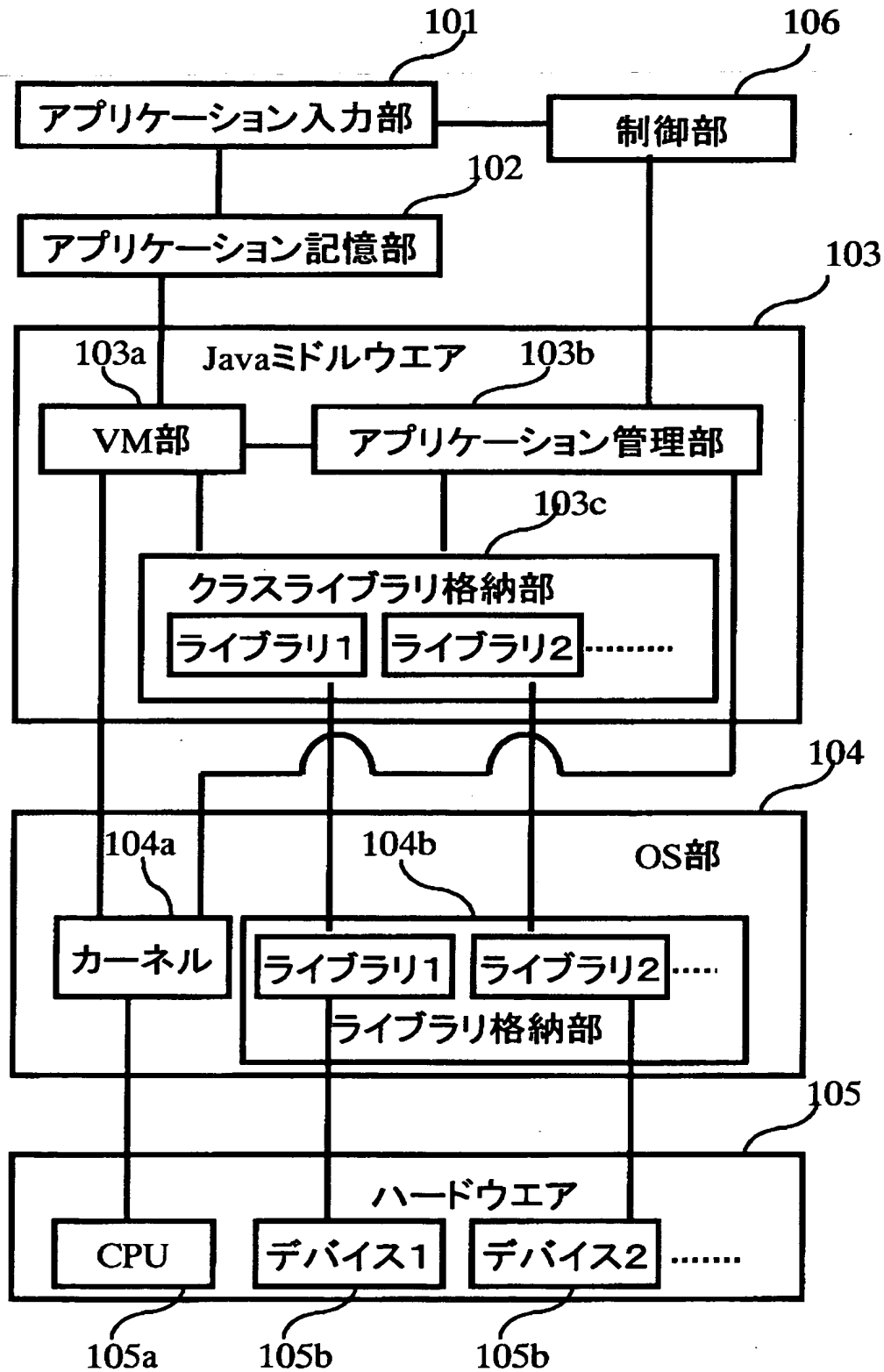
従来のアプリケーション実行装置が管理する資源の管理方法を示す図

【符号の説明】

- 1 0 1 アプリケーション入力部
- 1 0 2 アプリケーション記憶部
- 1 0 3 J a v a ミドルウェア
- 1 0 3 a VM部
- 1 0 3 b アプリケーション管理部
- 1 0 3 c クラスライブラリ格納部
- 1 0 4 O S
- 1 0 4 a カーネル
- 1 0 4 b ライブラリ格納部
- 1 0 5 ハードウェア
- 1 0 5 a C P U
- 1 0 5 b デバイス
- 1 0 6 制御部

【書類名】 図面

【図 1】



【図 2】

(1)

アプリケーションID	タスクID	Thread ID
1	201	1,2
2	202	4,5,6

(2)

アプリケーションID	タスクID	Thread ID
1	201	1,2,7
2	202	4,5,6

(3)

アプリケーションID	タスクID	Thread ID
1	201	1,2,7
2	202	4,6

【図 3】

(1)

アプリケーションID	スレッド インスタンス	リスナー インスタンス
1	thread10	L1,L4

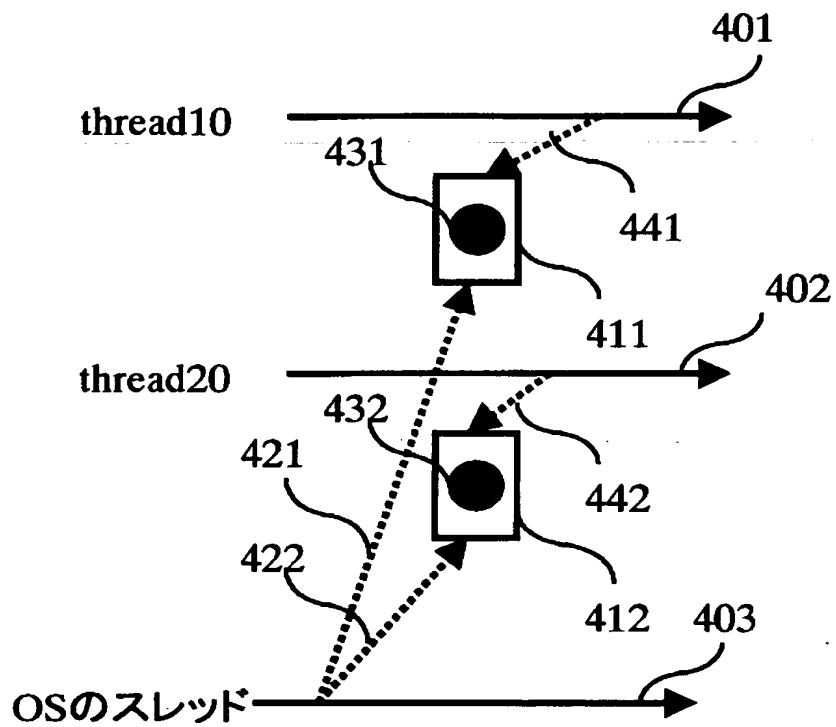
(2)

アプリケーションID	スレッド インスタンス	リスナー インスタンス
1	thread10	L1,L4
2	thread20	L6

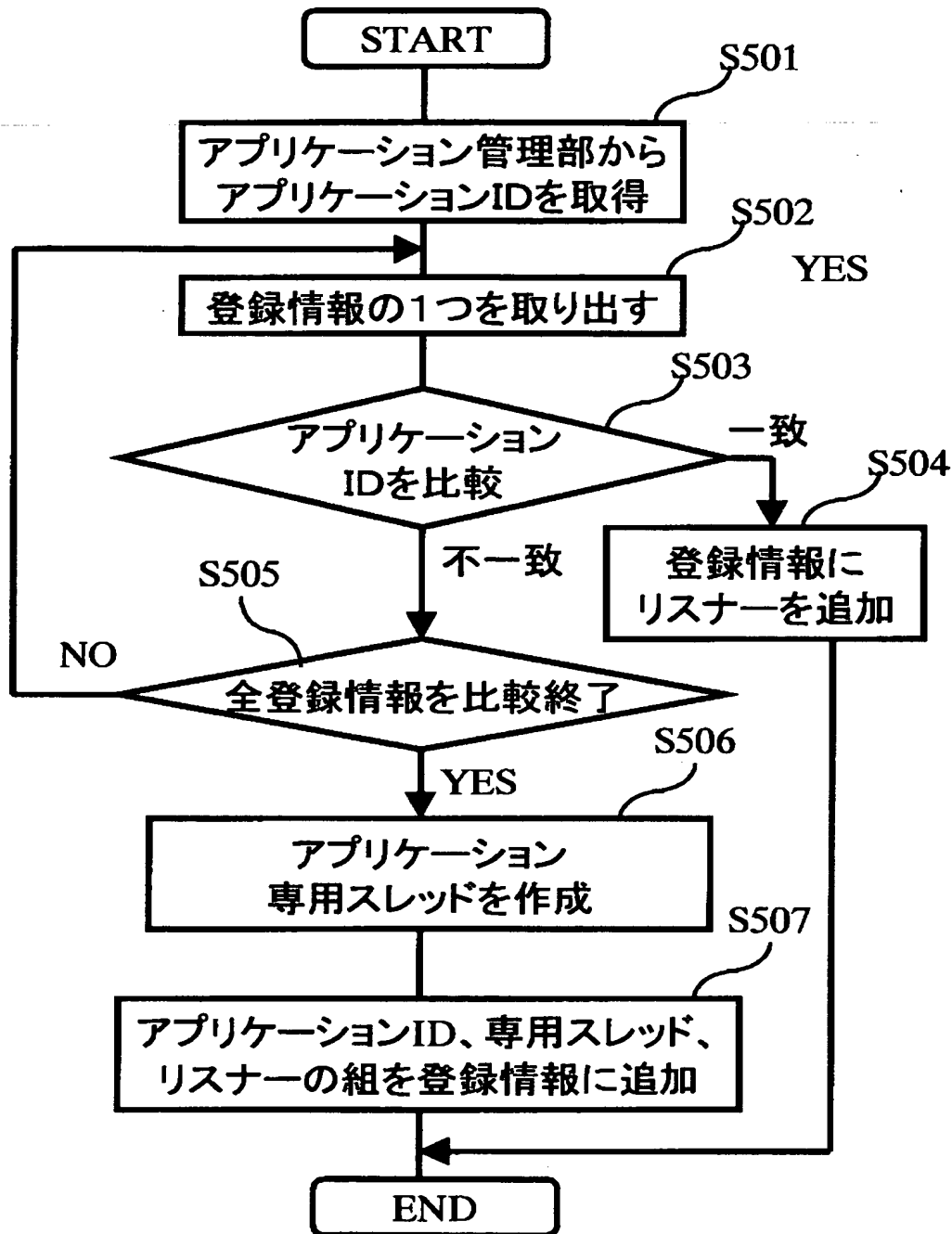
(3)

アプリケーションID	スレッド インスタンス	リスナー インスタンス
2	thread20	L6

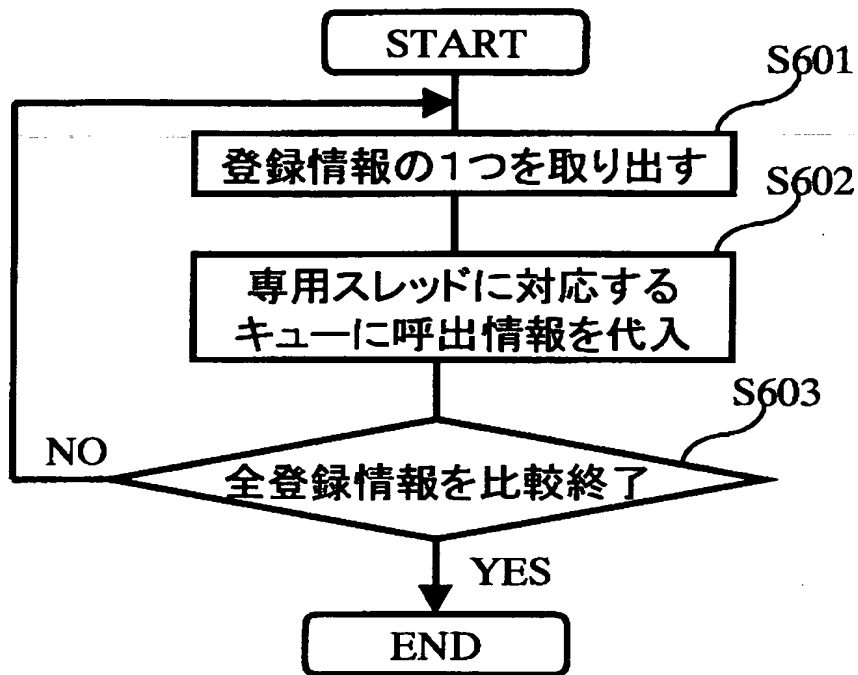
【図 4】



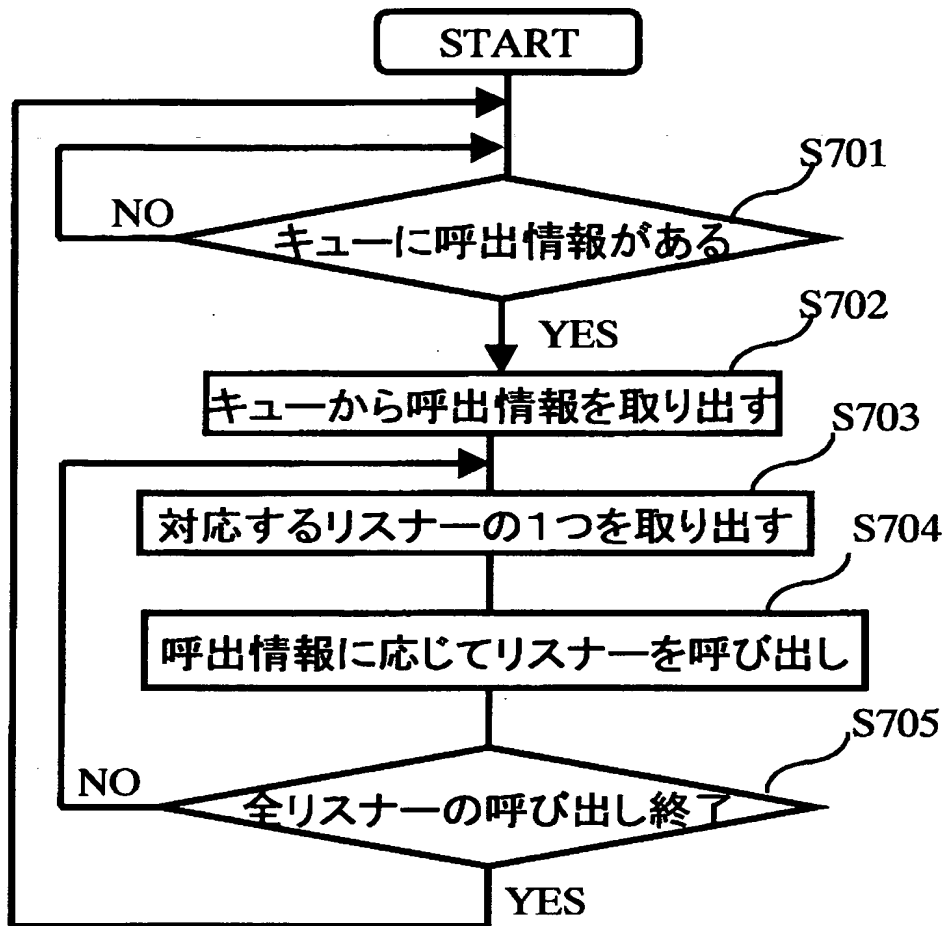
【図 5】



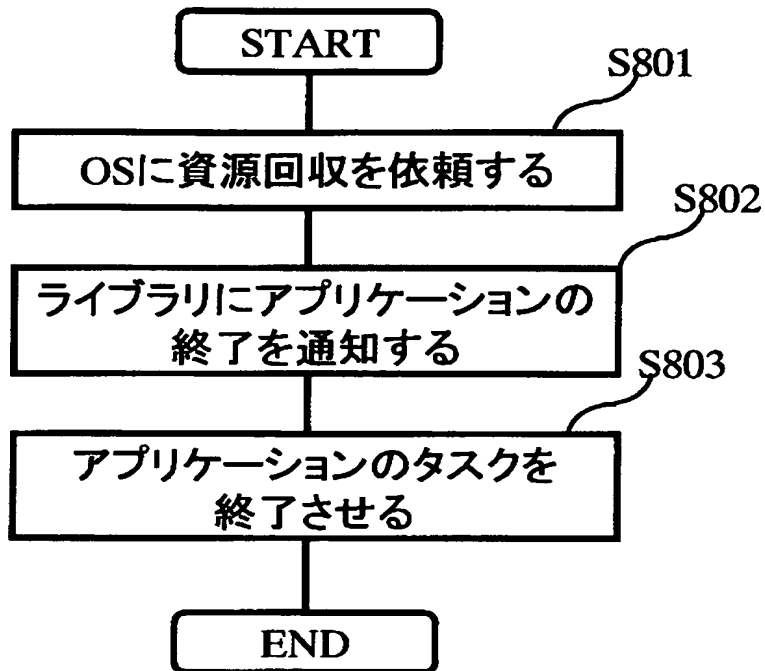
【図 6】



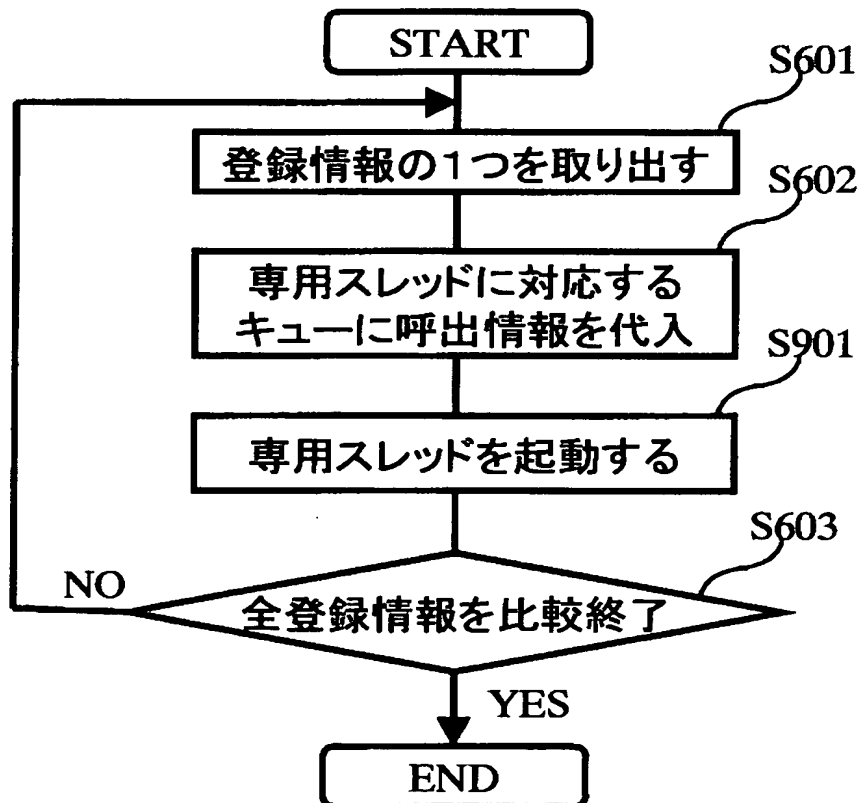
【図 7】



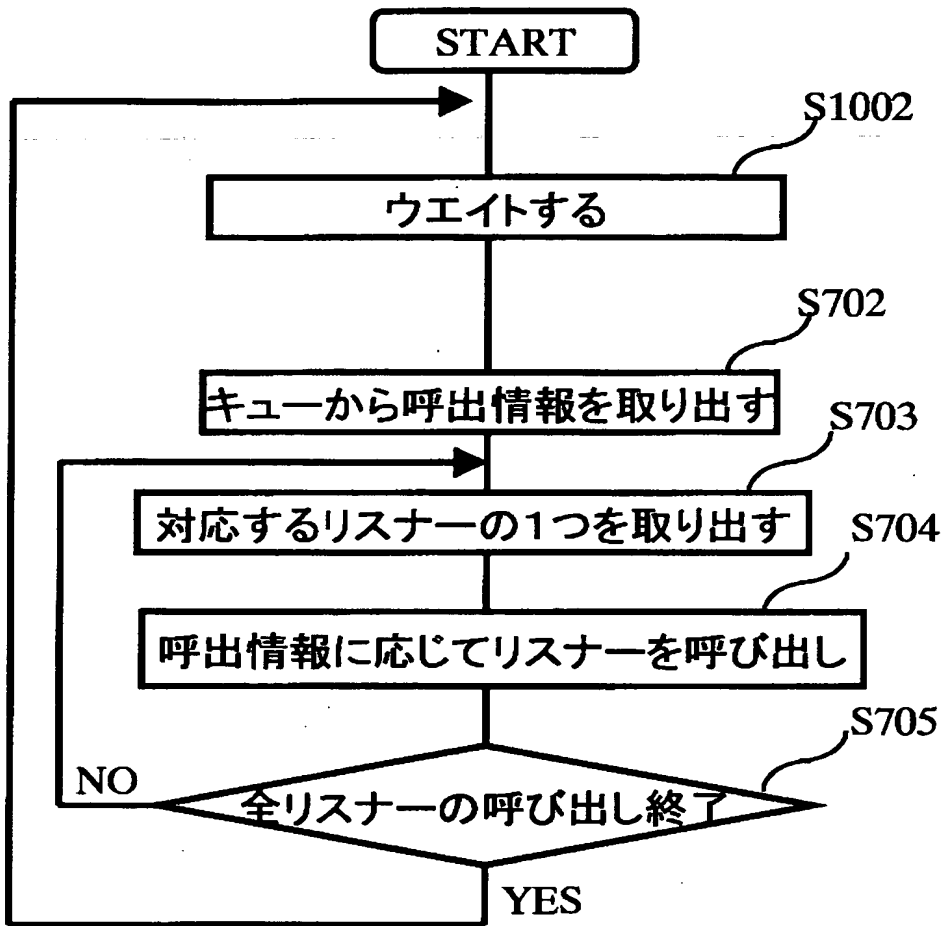
【図 8】



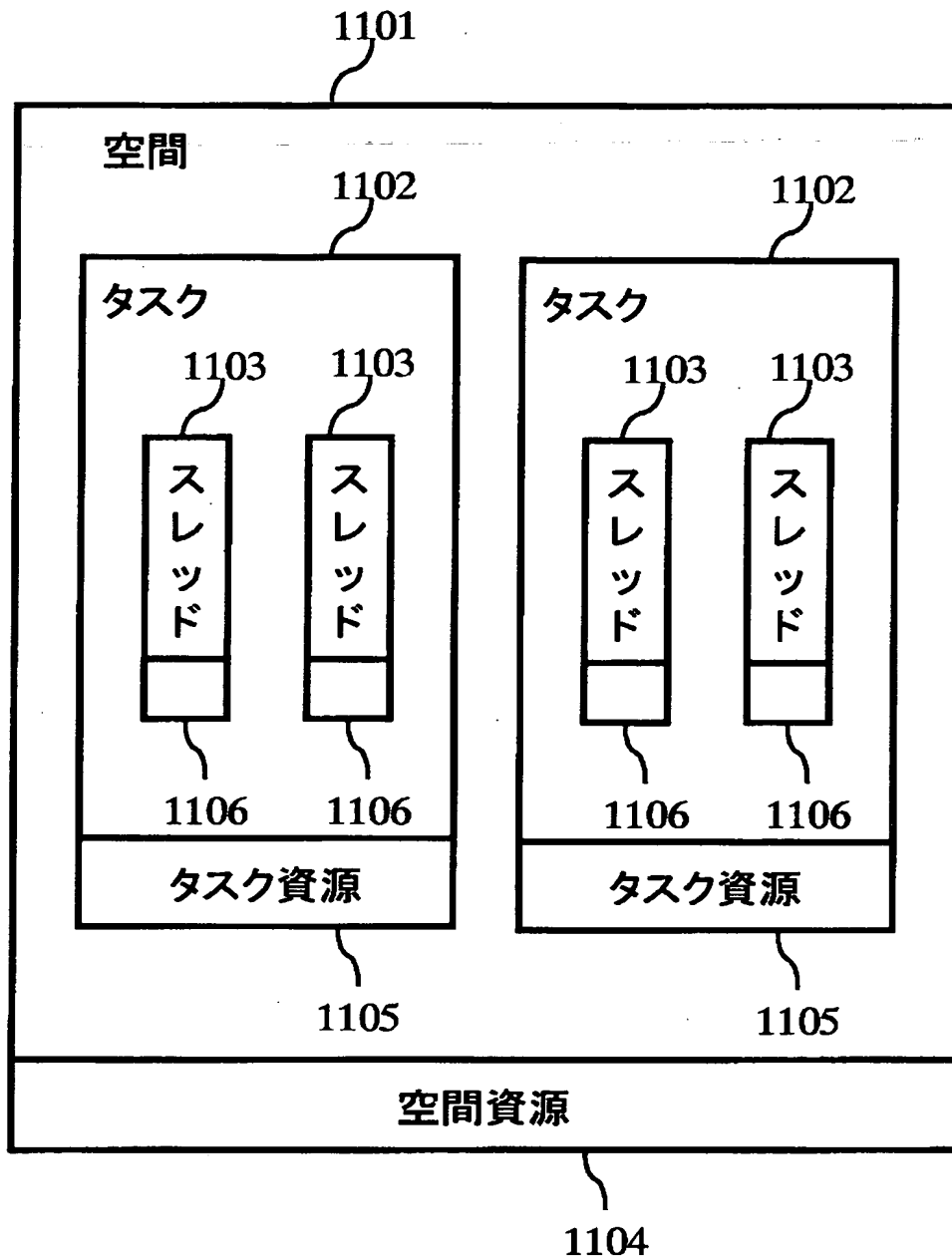
【図 9】



【図 1 0】



【図 1 1】



【書類名】 要約書

【要約】

【課題】 J a v a ミドルウェアを再起動せずに次のアプリケーションを実行すると資源回収が不十分なため、資源不足でアプリケーションが動作しなくなる。

【解決手段】 J a v a アプリケーション管理部 1 0 3 b はタスクとスレッドを O S 1 0 4 のカーネル 1 0 4 a に依頼して用意し、そのスレッド上で V M 部 1 0 3 a は J a v a アプリケーションを実行する。J a v a ミドルウェア 1 0 3 のクラスライブラリーは J a v a アプリケーションからリスナー登録される際、J a v a アプリケーションに対応するスレッドを生成する。O S 1 0 4 のライブラリーからリスナーを呼び出すタイミングを通知された時、クラスライブラリーは、生成しておいたスレッド上でリスナーを呼び出す。アプリケーション終了時、J a v a アプリケーション管理部 1 0 3 b はカーネル 1 0 4 a に依頼して J a v a アプリケーションが使用している資源を回収する。

【選択図】 図 1

出 願 人 履 歴 情 報

識別番号 [0 0 0 0 0 5 8 2 1]

1. 変更年月日	1 9 9 0 年 8 月 2 8 日
[変更理由]	新規登録
住 所	大阪府門真市大字門真 1 0 0 6 番地
氏 名	松下電器産業株式会社